A STUDY OF COMBINATORIAL ISSUES IN A SPARSE HYBRID SOLVER

ERIK G. BOMAN AND SIVASANKARAN RAJAMANICKAM

ABSTRACT. The solution of large sparse linear systems is an important kernel in scientific computing. Hybrid direct-iterative methods provide a compromise between the robustness of direct solvers and the lower memory requirements of iterative solvers. We show that combinatorial algorithms such as partitioning, ordering, and coloring play an important role in hybrid solvers. We identify combinatorial issues and study the effects of partitioning in a new hybrid solver, ShyLU.

1. INTRODUCTION

The solution of large, sparse systems of linear systems is an important part of many computations in computational science and engineering. There are two main types of solvers: direct and iterative. While direct solvers are very robust, they do not scale well. Iterative solvers are more scalable and better suited for parallel computing but are less robust. Recently, *hybrid* solvers have become popular for some applications. Hybrid solvers attempt to combine the robustness of direct solvers with the lower memory requirements and more parallel approach of iterative solvers. Examples of such hybrid solvers include HIPS [1] and PDSlin [2]. As part of a collaboration between the Combinatorial Scientific Computing and Petascale Simulations (CSCAPES) SciDAC institute and the Extreme-scale Algorithms and Software Institute (EASI), we have developed a new hybrid solver, ShyLU [3]. ShyLU is built on Trilinos and uses the Isorropia and Zoltan packages for partitioning and ordering of sparse matrices. We summarize the design of ShyLU and then discuss in detail some of the combinatorial issues that affect parallel performance.

2. SHYLU DESIGN

The Schur complement framework is a general way to solve linear systems. Much work has been done in this area; see, for example, [4, Ch.14] and the references therein.

Let $Ax = b$ be the system of interest. Suppose A has the form

$$(1) \qquad A = \begin{pmatrix} D & C \\ R & G \end{pmatrix},$$

where D and G are square and D is nonsingular. The Schur complement after elimination of the top row is $S = G - R * D^{-1}C$. Solving $Ax = b$ then consists of the three solves: $Dz = b_1$, $Sx_2 = b_2 - Rz$ and $Dx_1 = b_1 - Cx_2$ where the vector subscripts correspond to the matrix block rows.

The algorithms that use this formulation to solve the linear system in an iterative method or a hybrid method essentially use three basic steps. We call this the Schur complement framework.

2.1. **Partitioning.** The key idea is to permute A to get a D that is easy to factor. In this case, $D = diag(D_1, \ldots, D_k)$ is a block diagonal matrix, R is a row border, and C is a column border. For example, the symmetric case in Figure 1(b) is identical to the Schur complement formulation where we use a symmetric permutation PAP^T to get a doubly bordered block form. In the nonsymmetric case, we find the unsymmetric permutation PAQ, to get the singly bordered block diagonal form; see Figure 1(a). The nonsymmetric case can be solved by using the same Schur complement formulation even though it
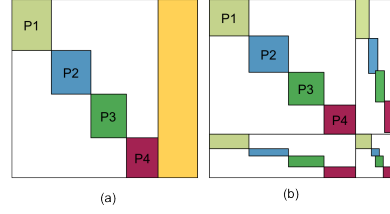
FIGURE 1.   Partitioning and reordering of a (a) nonsymmetric and (b) symmetric matrix.

appears different. Several variations of graph partitioning can be used to find the permutations and the block bordered structure. In Section 4 we compare the various options and their effects on the hybrid solver.

2.2. **Sparse Approximation of** S**.** Once D is factored (either exactly or inexactly), the crux of the Schur complement approach is to solve for S iteratively. S is typically much smaller than A and S is generally better conditioned than A. However, S is typically dense making it expensive to compute and store. All algorithms compute a sparse approximation of S to be used as a preconditioner either for an implicit S or for an inexact solve. We use Pardiso [5], a multithreaded sparse direct solver, for the blocks in D. ShyLU can use two methods to form $\bar{S} \approx S$: *dropping* and *probing*.

Dropping (value-based). With dropping we keep only the largest (in magnitude) entries of S. This is a common strategy and was also used in HIPS and PDSLin. When forming $S = G - R * D^{-1}C$, we simply drop entries less than a given threshold. We use a relative threshold, dropping entries that are smaller relative to the diagonal entries.

Probing (structure-based). Probing was developed to approximate interfaces in domain decomposition [6]. In probing, we prescribe the sparsity pattern of $\bar{S} \approx S$. Then we compute a set of probing vectors, V, based on $\bar{S}$. This gives rise to a coloring problem, where the number of colors correspond to the number of probing vectors needed. We then apply $S = G - RD^{-1}C$ as an operator to the probing vectors V to obtain SV, which gives us the numerical values for $\bar{S}$. Choosing the sparsity pattern of $\bar{S}$ can be tricky. For PDE problems where the values in S decay away from the diagonal, a band matrix is often used [6]. To strengthen our preconditioner, we include the pattern of G in the probing pattern. Thus the pattern of $\bar{S}$ is pattern of $B \cup G$, where B is a banded matrix. More details on probing can be found in [3]. We use the Isorropia package of Trilinos to compute the probing, which uses the Zoltan [7] for parallel graph coloring.

2.3. **Fast inexact solution with** S**.** Once we have an approximate S, there exist multiple options to solve using S and then solve for the entire system. A popular approach in hybrid methods is to solve the Schur complement system iteratively using $\bar{S}$ as a preconditioner. Since we need only an inexact solve as a preconditioner, we can also simply solve for $\bar{S}$ instead of for S. We solve for $\bar{S}$ iteratively in parallel, and we use yet another approximation $\tilde{S} \approx \bar{S}$ as a preconditioner for $\bar{S}$. In practice, $\tilde{S}$ can be simple, for example, Jacobi or block Jacobi. Once the preconditioner ($\bar{S}$ or $\tilde{S}$) and the operator for our solve (either an implicit S or $\bar{S}$) is decided, ShyLU uses the inner-outer iteration, with inner iteration for the Schur complement and the outer iteration for the whole system.

3. NARROW SEPARATORS VS WIDE SEPARATORS

The algorithm in Section 2 depends on finding separators to partition the matrix into the bordered form. Let (V_1, V_2, S) be a partition of the vertices V in a graph $G(V, E)$. S is a *separator* if there is no edge (v, w) such that $v \in V_1$ and $w \in V_2$. Separator S is called a *wide separator* if any path from V_1 to V_2 contains at least two vertices in S. A separator that is not wide is called a *narrow separator*. Note that the edge separator as computed by many of the partitioning packages is a wide separator.

Wide separators were originally used as part of ordering techniques for sparse Gaussian elimination [8]. The intended application at that time was sparse direct factorization [9]. We revisit this comparison with respect to hybrid solvers here.

The doubly bordered block diagonal form of a matrix A when we use a narrow separator is shown below (for two parts).

$$
(2) \qquad A_{narrow} = \begin{pmatrix} D_{11} & 0 & C_{11} & C_{12} \\ 0 & D_{22} & C_{21} & C_{22} \\ R_{11} & R_{12} & G_{11} & G_{12} \\ R_{21} & R_{22} & G_{21} & G_{22} \end{pmatrix}
$$

All the R_{ij} blocks and C_{ij} blocks can have nonzeros in them. As a result, every block in the Schur complement might require communication when we compute it. For example, while using the matrix from the narrow separator A_{narrow}, to compute the S_{11} block of the Schur complement, we do the following.

$$
(3) \qquad S_{11} = G_{11} - R_{11} * D_1^{-1} * C_{11} + R_{12} * D_2^{-1} * C_{21}
$$

Computing the Schur complement in the above form is expensive because of the communication involved. However, the doubly bordered block diagonal form for two parts when we use a wide separator has more structure to it, as shown below.

$$
(4) \qquad A_{wide} = \begin{pmatrix} D_{11} & 0 & C_{11} & 0 \\ 0 & D_{22} & 0 & C_{22} \\ R_{11} & 0 & S_{11} & S_{12} \\ 0 & R_{22} & S_{21} & S_{22} \end{pmatrix}
$$

Consider that rows of D_{ij} are the interior vertices in part i and the rows in R_{ij} are boundary vertices in part i then we observe that all blocks R_{ij} and C_{ij} will be equal to zero when $i \neq j$. This follows from the definition of the wide separator.

As R and C are block diagonal matrices, we can compute the Schur complement without any communication. For example, to compute the S_{11} block of the Schur complement of A_{wide} we do the following.

$$
(5) \qquad S_{11} = G_{11} - R_{11} * D_1^{-1} * C_{11}
$$

Thus, computing S in the wide separator case is fully parallel. The off-diagonal blocks of the Schur complement are equal to the off-diagonal blocks of G. However, the wide separator can be as much as two times the size of the narrow separator. This results in a larger Schur complement system to be solved when using the wide separator. In hybrid solvers, we solve the Schur complement system in parallel as well. As a result, while the bigger Schur complement system leads to increased solve time, the much faster setup due to increased parallelism offsets the small increase in solve time.

4. PARTITIONING METRICS

The algorithm described in Section 2 depends on finding a "good" partition. As a result, the performance of the hybrid solver depends on the partitioning as well. Both hypergraph partitioning and graph partitioning can be used to compute the wide separators described in Section 3. Though partitioning algorithms are designed with iterative solvers as their target to optimize, these are the closest tools available today. They are not too far apart in what they try to achieve. The load balance in each subdomain is important as that is the amount of work in the direct factorization. However, in terms of the metric they choose to minimize different partitioning algorithms help the hybrid solver in different ways. We compare three options.

Hypergraph partitioning in Zoltan can use two different metrics.

TABLE 1.   Comparison of number of iterations and solve time of ShyLU for different partitioning metrics

Matrix Name	Method Method	Rows in G	Outer Iter	Solve Time	Inner LB
wathen240K	Graph	**3940**	10	2.61	1.10
	Cutnet	5641	9	**2.21**	1.23
	lambda	4201	9	2.31	1.22
bodyy5	Graph	577	59	0.68	1.08
	Cutnet	**523**	55	0.64	1.05
	lambda	533	55	0.644	1.05
Pres_Poisson	Graph	**1248**	46	1.61	1.34
	Cutnet	1472	40	1.63	1.42
	lambda	1816	90	3.82	1.45
Lourakis	Graph	**3267**	19	0.35	1.40
	Cutnet	3279	19	0.68	**2.21**
	lambda	3300	18	0.608	**1.94**
venkat50	Graph	**1468**	129	13.12	1.32
	Cutnet	1608	170	16.87	1.18
	lambda	1756	143	15.93	1.52
Xyce_1	Graph	*			
	Cutnet	10330	1	0.15	1.64
	lambda	**9712**	1	0.15	1.17
ckt11752_dc_1	Graph	957	214	9.88	1.46
	Cutnet	650	*	*	2.70
	lambda	**638**	*	*	2.52
memplus	Graph	4917	357	7.75	1.33
	Cutnet	**4174**	349	8.64	1.90
	lambda	4604	299	**6.43**	1.38

- $\lambda - 1$ metric, which minimize the communication volume in the matrix vector multiplication.
- cutnet metric, which minimizes the number of cut hyperedges in the original hypegraph.

In terms of the hybrid solvers, the $\lambda - 1$ metric is useful because it minimizes the communication volume in matrix-vector multiply of the outer iteration. In contrast, the cutnet metric actually minimizes the number of rows in the Schur complement. The exact description of the metrics can be found at [7]. We could also use graph partitioning, which minimizes edge-cut metric. The edge cut metric minimizes the number of off-diagonal entries in the G block.

In contrast, the following metric would be most useful for the hybrid solver:

- Minimize the communication volume in the outer iteration (as in the $\lambda - 1$ metric).
- Minimize the communication volume in the inner iteration.
- Balance the work in the inner iteration (or the number of boundary vertices).
- Minimize number of rows in G or the Schur complement (as given by the cutnet metric).

As none of the available tools minimize the multiple constraints given above, we study what is the best approximation available for what is required by the solver. We compare the $\lambda - 1$ metric, the cutnet metric, and the edge cut metric with respect to what is important for the hybrid solver—fewer outer iterations and a better solve time.

We use matrices from the University of Florida sparse matrix collection as well as matrices from our applications for these tests. All the tests were run on a desktop with four MPI tasks and one thread for the direct solver. The results are shown in Table 1. We use Zoltan's parallel hypergraph partitioning to compute the hypergraph partitioning metrics and ParMETIS to compute the graph partitioning.

Some of the important observations from our test runs are as follows.

- No one metric works best for all the problems when the target usage is in hybrid solvers.
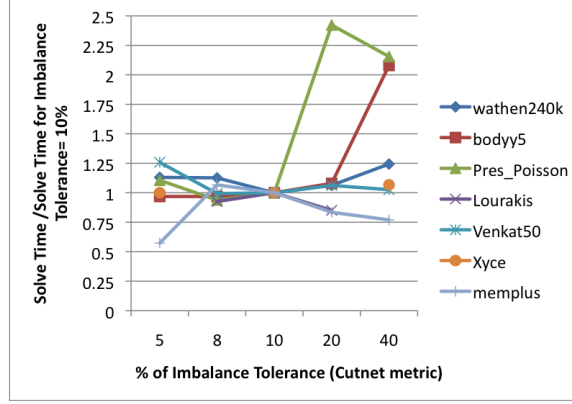
FIGURE 2. Comparison of solve time of ShyLU for various values of imbalance tolerance with the cutnet metric. The values are normalized for the solve time when imbalance tolerance = 10%.

- The smallest Schur complement does not necessarily lead to best solve time (for example, see the solve time for the matrix wathen240k).
- The load imbalance in the inner iteration can adversely affect the solve time even when the Schur complement sizes are small (for example, both cutnet and $\lambda - 1$ metrics give smaller Schur complement for the matrix Lourakis, but result in poor load balance for inner iteration, and poor solve time.
- The smallest Schur complement need not even converge (for example, see the results for the matrix ckt11752_dc_1).

As shown in Table 1, load imbalance in the inner iteration is important for better solver performance. While most partitioning software allows a tolerance to be set for imbalance among different parts, there is no recommended tolerance for hybrid solvers. The defaults vary a lot from one partitioner to the other. In this subsection, we use the same matrices as before, in order to study the effect of how load imbalance affects the size of the Schur complement and in turn the solver time.

As the load imbalance tolerance is relaxed, the Schur complement should get smaller; however, the load imbalance will affect the triangular solves in each part will not be balanced and will adversely affect the solve time. The results are shown in Figure 2. We normalize the results to the solve time when the imbalance tolerance percentage is 10% (default in Zoltan). For most matrices an imbalance tolerance of 8–10% is the best. There is one matrix, memplus, where 5% imbalance tolerance does better, and another matrix, Lourakis, where 40% imbalance tolerance does better.

5. CONCLUSION

We studied some of the combinatorial problems in hybrid solvers such as ShyLU. Parallel graph coloring algorithms enabled us to do robust probing for hybrid solvers, which is implemented in ShyLU. We argue that in a parallel context, especially with respect to hybrid solvers, wide separators have an important role to play. We also compared different partitioning metrics with respect to solver performance. No one partitioning metrics works well for all our test problems. More work is needed to use a multiconstraint graph or hypergraph partitioning for the objectives of a hybrid solver. We also studied what will be a good imbalance tolerance for partitioning tools. Our preliminary study shows 8–10% is a good imbalance tolerance. A study of any of these combinatorial problems will help other applications in computational science as well.

REFERENCES

[1] J. Gaidamour and P. Henon. A parallel direct/iterative solver based on a Schur complement approach. *Computational Science and Engineering, IEEE International Conference on*, 0:98–105, 2008.

[2] Ichitaro Yamazaki and Xiaoye S. Li. On techniques to improve robustness and scalability of a parallel hybrid linear solver. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, VECPAR'10, pages 421–434, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Sivasankaran Rajamanickam, Erik G. Boman, and Heroux Michael A. A hybrid-hybrid solver for manycore platforms. Submitted.

[4] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.

[5] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.

[6] Tony F. C. Chan and Tarek P. Mathew. The interface probing technique in domain decomposition. *SIAM J. Matrix Anal. Appl.*, 13:212–238, January 1992.

[7] K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, and U.V. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.

[8] John R. Gilbert and Earl Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, 16:427–449, 1987. 10.1007/BF01388998.

[9] Alan George, Michael T. Heath, Joseph Liu, and Esmond NG. Sparse Cholesky factorization on a local-memory multiprocessor. *SIAM J. Sci. Stat. Comput.*, 9:327–340, March 1988.